Autocomplete

Autocomplete A powerful and flexible autocomplete input component that provides intelligent search and selection capabilities. Features include single and multi-select modes, custom option templates, loading states, keyboard navigation, and comprehensive accessibility support. Perfect for search interfaces, form inputs, and data selection workflows. How to use ■ import { AvaAutocompleteComponent } from "@aava/play-core" ; Basic Usage ■ The most basic implementation with a simple array of options and default single-select behavior. Multi-Select Mode ■ Enable multi-select functionality to allow users to select multiple options, displayed as removable tags. Multi-Select Features ■ Tag Display : Selected options appear as removable tags below the input Tag Customization : Full control over tag appearance, colors, and behavior Duplicate Prevention : Automatically prevents duplicate selections Bulk Removal : Individual tag removal with click or keyboard Clear All : Option to clear all selections at once Options with Icons ■ Enhance autocomplete options with meaningful icons for better visual context and user recognition. Icon Features ■ Option Icons : Add icons to individual options for visual distinction Start Icon : Include a leading icon in the input field itself Icon Customization : Control icon colors, sizes, and positioning Consistent Theming : Icons inherit theme colors automatically Accessibility : Proper ARIA labels for icon descriptions Async States ■ Display loading indicators while fetching options from asynchronous sources. Async Features ■ Debounced Requests : Configurable debounce to prevent excessive API calls Loading Text : Customizable loading message State Management : Automatic loading state handling Error Handling : Graceful error state display Accessibility ■ Built-in accessibility features ensuring WCAG compliance and inclusive user experience. Accessibility Features ■ Keyboard Navigation : Full arrow key, Enter, Escape, and Tab support ARIA Attributes : Proper aria-autocomplete , aria-controls , and aria-activedescendant Screen Reader Support : Descriptive labels and state announcements Focus Management : Clear visual focus indicators High Contrast : Enhanced visibility in high contrast mode Reduced Motion : Respects user motion preferences API Reference ■ Inputs ■ Property Type Default Description options AvaAutocompleteOption[] | Observable<AvaAutocompleteOption[]> [] Array or Observable of autocomplete options placeholder string '' Placeholder text for the input field label string '' Label for the input field error string '' Error message to display helper string '' Helper text below the input loading boolean false Show loading state disabled boolean false Disable the autocomplete clearable boolean true Show clear button when input has value minLength number 1 Minimum characters to trigger search maxOptions number 10 Maximum number of options to display noResultsText string 'No results found' Text shown when no options match debounce number 200 Debounce time for input changes (ms) optionTemplate TemplateRef<unknown> undefined Custom template for option display multi boolean false Enable multi-select mode fullWidth boolean false Make the component full width required boolean false Mark field as required readonly boolean false Make input readonly name string '' Name attribute for the input id string '' ID attribute for the input ariaLabel string '' ARIA label for accessibility ariaLabelledby string '' ARIA labelledby

attribute ariaDescribedby string '' ARIA describedby attribute Tag Properties (Multi-Select) ■ Property Type Default Description tagColor 'default' | 'primary' | 'success' | 'warning' | 'error' | 'info' | 'custom' 'default' Color theme for selected tags tagVariant 'filled' | 'outlined' 'filled' Visual variant for tags tagSize 'sm' | 'md' | 'lg' 'sm' Size of the tags tagPill boolean false Use pill shape for tags tagRemovable boolean true Allow tags to be removed tagDisabled boolean false Disable tag interactions tagIcon string '' Icon to display in tags tagIconPosition 'start' | 'end' 'start' Position of icon in tags tagAvatar string '' Avatar image URL for tags tagCustomStyle Record<string, string> {} Custom styles for tags tagCustomClass string '' Custom CSS class for tags tagIconColor string '' Custom color for tag icons Icon Properties ■ Property Type Default Description startIcon string '' Icon name for the start of the input startIconColor string '' Color for the start icon startIconSize string '16px' Size of the start icon Outputs ■ Event Type Description optionSelected EventEmitter<AvaAutocompleteOption> Emitted when an option is selected valueChange EventEmitter<string | string[]> Emitted when the input value changes cleared EventEmitter<void> Emitted when the input is cleared AvaAutocompleteOption Interface ■ interface AvaAutocompleteOption { label : string ; // Display text for the option value : string ; // Value to be emitted when selected icon ? : string ; // Optional icon name for the option group ? : string ; // Optional group for categorization [key : string] : string | boolean | number | undefined ; // Additional custom properties } Methods ■ Method Parameters Return Type Description onInput(event: Event) event: Event void Handle input changes and trigger search onFocus() - void Handle focus events onBlur() - void Handle blur events onOptionClick(option: AvaAutocompleteOption) option: AvaAutocompleteOption void Handle option selection onClear() - void Clear the input and selections onKeydown(event: KeyboardEvent) event: KeyboardEvent void Handle keyboard navigation removeSelectedOption(opt: AvaAutocompleteOption) opt: AvaAutocompleteOption void Remove a selected option (multi-select) CSS Custom Properties ■ Property Default Description --autocomplete-background Dynamic Background color for dropdown --autocomplete-border-color Dynamic Border color for dropdown --autocomplete-border-width Dynamic Border width for dropdown --autocomplete-border-radius Dynamic Border radius for dropdown --autocomplete-shadow Dynamic Box shadow for dropdown --autocomplete-option-color Dynamic Text color for dropdown options --autocomplete-option-hover-bg Dynamic Background color for hovered options --autocomplete-option-hover-color Dynamic Text color for hovered options --autocomplete-option-icon-color Dynamic Color for option icons --autocomplete-empty-color Dynamic Color for "no results" text --autocomplete-chip-bg Dynamic Background color for multi-select tags --autocomplete-chip-color Dynamic Text color for multi-select tags --autocomplete-chip-remove-color Dynamic Color for tag remove buttons Accessibility Guidelines ■ Keyboard Navigation ■ Tab : Navigate to autocomplete and move between interactive elements Arrow Down/Up : Navigate through dropdown options Enter : Select highlighted option Escape : Close dropdown without selection Backspace : In multi-select, remove last selected option Space : Select highlighted option (alternative to Enter) Screen Reader Support ■ Use descriptive labels that clearly indicate the autocomplete purpose Provide context about the total number of options

available Announce option changes and selection updates Include loading state announcements Use appropriate ARIA attributes for state communication Visual Design ■ Maintain sufficient color contrast (4.5:1 minimum) for all states Provide clear focus indicators on interactive elements Ensure dropdown options meet minimum touch target size (44px) Use consistent visual hierarchy across all states Support high contrast and reduced motion preferences Best Practices ■ Design Guidelines ■ Clear Labels : Use descriptive labels that explain the autocomplete purpose Appropriate Placeholders : Provide helpful placeholder text with examples Option Grouping : Group related options when dealing with large datasets Loading Feedback : Always show loading states for async operations Error Handling : Provide clear error messages and recovery options Responsive Design : Ensure dropdown adapts to different screen sizes Performance ■ Debouncing : Use appropriate debounce times to prevent excessive API calls Option Limiting : Limit displayed options to prevent performance issues Lazy Loading : Consider lazy loading for very large option sets Memory Management : Clean up subscriptions and event listeners Virtual Scrolling : Implement virtual scrolling for thousands of options Caching : Cache frequently accessed options to improve response times Form Integration ■ Validation : Implement proper form validation for required fields Error States : Display validation errors clearly and consistently Default Values : Handle default values appropriately in both single and multi-select Form Submission : Ensure selected values are properly included in form data Accessibility : Maintain proper form accessibility throughout the component lifecycle Multi-Select Considerations ■ Tag Management : Provide clear ways to remove individual tags Bulk Operations : Consider bulk remove functionality for many selections Tag Overflow : Handle cases where many tags exceed available space Keyboard Navigation : Ensure proper keyboard navigation through tags Visual Feedback : Provide clear visual feedback for tag interactions

```
import { Component } from "@angular/core";
import {
  AvaAutocompleteComponent,
  AvaAutocompleteOption,
} from "@aava/play-core";

@Component({
  selector: "app-autocomplete-basic",
  standalone: true,
  imports: [AvaAutocompleteComponent],
  template: `
    <div class="demo-container">
      <h3>Basic Autocomplete</h3>

      <ava-autocomplete
        [options]="countries"
        placeholder="Search for a country..."
        label="Country"
        (optionSelected)="onOptionSelected($event)"
        (valueChange)="onValueChange($event)"
      ></ava-autocomplete>

      <div class="demo-output" *ngIf="selectedCountry">
        <p><strong>Selected:</strong> {{ selectedCountry }}</p>
      </div>
    </div>
  `,
  styles: [
    `
      .demo-container {
        max-width: 400px;
        margin: 20px 0;
      }

      .demo-output {
        margin-top: 20px;
        padding: 12px;
        background: #f8f9fa;
        border-radius: 6px;
        border-left: 4px solid #007bff;
      }
    `,
  ],
})
export class AutocompleteBasicDemo {
  countries: AvaAutocompleteOption[] = [
    { label: "United States", value: "us" },
    { label: "Canada", value: "ca" },
    { label: "United Kingdom", value: "uk" },
    { label: "Germany", value: "de" },
    { label: "France", value: "fr" },
    { label: "Italy", value: "it" },
    { label: "Spain", value: "es" },
    { label: "Netherlands", value: "nl" },
    { label: "Belgium", value: "be" },
    { label: "Switzerland", value: "ch" },
```

```
    { label: "Austria", value: "at" },
    { label: "Sweden", value: "se" },
    { label: "Norway", value: "no" },
    { label: "Denmark", value: "dk" },
    { label: "Finland", value: "fi" },
  ];

  selectedCountry: string = "";

  onOptionSelected(option: AvaAutocompleteOption) {
    console.log("Option selected:", option);
  }

  onValueChange(value: string) {
    console.log("Value changed:", value);
    this.selectedCountry = value;
  }
}
```

```
import { Component } from "@angular/core";
import {
  AvaAutocompleteComponent,
  AvaAutocompleteOption,
} from "@aava/play-core";

@Component({
  selector: "app-autocomplete-multi-select",
  standalone: true,
  imports: [AvaAutocompleteComponent],
  template: `
    <div class="demo-container">
      <h3>Multi-Select Autocomplete</h3>

      <ava-autocomplete
        [options]="skills"
        [multi]="true"
        placeholder="Search and select skills..."
        label="Skills"
        tagColor="primary"
        tagVariant="filled"
        tagSize="sm"
        tagPill="true"
        tagRemovable="true"
        (optionSelected)="onOptionSelected($event)"
        (valueChange)="onValueChange($event)"
      ></ava-autocomplete>

      <div class="demo-output" *ngIf="selectedSkills.length">
        <p><strong>Selected Skills:</strong></p>
        <ul>
          <li *ngFor="let skill of selectedSkills">{{ skill }}</li>
        </ul>
      </div>
    </div>
  `,
  styles: [
    `
      .demo-container {
        max-width: 500px;
        margin: 20px 0;
      }

      .demo-output {
        margin-top: 20px;
        padding: 12px;
        background: #f8f9fa;
        border-radius: 6px;
        border-left: 4px solid #28a745;
      }

      .demo-output ul {
        margin: 8px 0;
        padding-left: 20px;
      }
```

```
      .demo-output li {
        margin: 4px 0;
      }
    `,
  ],
})
export class AutocompleteMultiSelectDemo {
  skills: AvaAutocompleteOption[] = [
    { label: "Angular", value: "angular" },
    { label: "React", value: "react" },
    { label: "Vue.js", value: "vue" },
    { label: "TypeScript", value: "typescript" },
    { label: "JavaScript", value: "javascript" },
    { label: "HTML5", value: "html5" },
    { label: "CSS3", value: "css3" },
    { label: "Sass", value: "sass" },
    { label: "Node.js", value: "nodejs" },
    { label: "Python", value: "python" },
    { label: "Java", value: "java" },
    { label: "C#", value: "csharp" },
    { label: "PHP", value: "php" },
    { label: "Ruby", value: "ruby" },
    { label: "Go", value: "go" },
    { label: "Rust", value: "rust" },
    { label: "Docker", value: "docker" },
    { label: "Kubernetes", value: "kubernetes" },
    { label: "AWS", value: "aws" },
    { label: "Azure", value: "azure" },
    { label: "Google Cloud", value: "gcp" },
    { label: "MongoDB", value: "mongodb" },
    { label: "PostgreSQL", value: "postgresql" },
    { label: "MySQL", value: "mysql" },
    { label: "Redis", value: "redis" },
  ];

  selectedSkills: string[] = [];

  onOptionSelected(option: AvaAutocompleteOption) {
    console.log("Option selected:", option);
  }

  onValueChange(values: string[]) {
    console.log("Values changed:", values);
    this.selectedSkills = values;
  }
}
```

```
import { Component } from "@angular/core";
import {
  AvaAutocompleteComponent,
  AvaAutocompleteOption,
} from "@aava/play-core";

@Component({
  selector: "app-autocomplete-icons",
  standalone: true,
  imports: [AvaAutocompleteComponent],
  template: `
    <div class="demo-container">
      <h3>Autocomplete with Icons</h3>

      <ava-autocomplete
        [options]="socialPlatforms"
        placeholder="Search social platforms..."
        label="Social Platform"
        startIcon="search"
        startIconColor="#6b7280"
        startIconSize="18px"
        (optionSelected)="onOptionSelected($event)"
        (valueChange)="onValueChange($event)"
      ></ava-autocomplete>

      <div class="demo-output" *ngIf="selectedPlatform">
        <p><strong>Selected Platform:</strong> {{ selectedPlatform }}</p>
      </div>
    </div>
  `,
  styles: [
    `
      .demo-container {
        max-width: 400px;
        margin: 20px 0;
      }

      .demo-output {
        margin-top: 20px;
        padding: 12px;
        background: #f8f9fa;
        border-radius: 6px;
        border-left: 4px solid #ff6b35;
      }
    `,
  ],
})
export class AutocompleteIconsDemo {
  socialPlatforms: AvaAutocompleteOption[] = [
    { label: "Facebook", value: "facebook", icon: "facebook" },
    { label: "Twitter", value: "twitter", icon: "twitter" },
    { label: "Instagram", value: "instagram", icon: "instagram" },
    { label: "LinkedIn", value: "linkedin", icon: "linkedin" },
    { label: "YouTube", value: "youtube", icon: "youtube" },
    { label: "TikTok", value: "tiktok", icon: "music" },
    { label: "Snapchat", value: "snapchat", icon: "camera" },
```

```
    { label: "Pinterest", value: "pinterest", icon: "heart" },
    { label: "Reddit", value: "reddit", icon: "message-circle" },
    { label: "Discord", value: "discord", icon: "message-square" },
    { label: "Slack", value: "slack", icon: "message-circle" },
    { label: "WhatsApp", value: "whatsapp", icon: "phone" },
    { label: "Telegram", value: "telegram", icon: "send" },
    { label: "GitHub", value: "github", icon: "github" },
    { label: "Stack Overflow", value: "stackoverflow", icon: "help-circle" },
  ];

  selectedPlatform: string = "";

  onOptionSelected(option: AvaAutocompleteOption) {
    console.log("Option selected:", option);
  }

  onValueChange(value: string) {
    console.log("Value changed:", value);
    this.selectedPlatform = value;
  }
}
```

```
import { Component, OnInit } from "@angular/core";
import {
  AvaAutocompleteComponent,
  AvaAutocompleteOption,
} from "@aava/play-core";
import { Observable, of, delay } from "rxjs";

@Component({
  selector: "app-autocomplete-loading",
  standalone: true,
  imports: [AvaAutocompleteComponent],
  template: `
    <div class="demo-container">
      <h3>Autocomplete with Loading States</h3>

      <ava-autocomplete
        [options]="asyncOptions"
        [loading]="isLoading"
        placeholder="Search users (simulated API delay)..."
        label="Users"
        [debounce]="300"
        [minLength]="2"
        (optionSelected)="onOptionSelected($event)"
        (valueChange)="onValueChange($event)"
      ></ava-autocomplete>

      <div class="demo-output" *ngIf="selectedUser">
        <p><strong>Selected User:</strong> {{ selectedUser }}</p>
      </div>

      <div class="demo-info">
        <p>
          <small
            >■ Type at least 2 characters to trigger the simulated API call
            with loading state.</small
          >
        </p>
      </div>
    </div>
  `,
  styles: [
    `
      .demo-container {
        max-width: 400px;
        margin: 20px 0;
      }

      .demo-output {
        margin-top: 20px;
        padding: 12px;
        background: #f8f9fa;
        border-radius: 6px;
        border-left: 4px solid #17a2b8;
      }

      .demo-info {
```

```
        margin-top: 16px;
        padding: 8px 12px;
        background: #e7f3ff;
        border-radius: 4px;
        border-left: 3px solid #007bff;
      }

      .demo-info p {
        margin: 0;
        color: #0056b3;
      }
    `,
  ],
})
export class AutocompleteLoadingDemo implements OnInit {
  asyncOptions: Observable<AvaAutocompleteOption[]> = of([]);
  isLoading = false;
  selectedUser: string = "";

  private allUsers: AvaAutocompleteOption[] = [
    { label: "John Doe", value: "john.doe@example.com" },
    { label: "Jane Smith", value: "jane.smith@example.com" },
    { label: "Mike Johnson", value: "mike.johnson@example.com" },
    { label: "Sarah Wilson", value: "sarah.wilson@example.com" },
    { label: "David Brown", value: "david.brown@example.com" },
    { label: "Emily Davis", value: "emily.davis@example.com" },
    { label: "Michael Miller", value: "michael.miller@example.com" },
    { label: "Lisa Garcia", value: "lisa.garcia@example.com" },
    { label: "Robert Martinez", value: "robert.martinez@example.com" },
    { label: "Jennifer Anderson", value: "jennifer.anderson@example.com" },
    { label: "William Taylor", value: "william.taylor@example.com" },
    { label: "Amanda Thomas", value: "amanda.thomas@example.com" },
    { label: "James Jackson", value: "james.jackson@example.com" },
    { label: "Michelle White", value: "michelle.white@example.com" },
    { label: "Christopher Harris", value: "christopher.harris@example.com" },
  ];

  ngOnInit() {
    // Simulate async options with loading state
    this.asyncOptions = new Observable((observer) => {
      observer.next([]);
    });
  }

  onOptionSelected(option: AvaAutocompleteOption) {
    console.log("Option selected:", option);
  }

  onValueChange(value: string) {
    console.log("Value changed:", value);
    this.selectedUser = value;

    // Simulate API call with loading state
    if (value.length >= 2) {
      this.isLoading = true;
```

```
      // Simulate API delay
      setTimeout(() => {
        const filteredUsers = this.allUsers.filter((user) =>
          user.label.toLowerCase().includes(value.toLowerCase())
        );

        this.asyncOptions = of(filteredUsers).pipe(delay(500));
        this.isLoading = false;
      }, 300);
    } else {
      this.asyncOptions = of([]);
    }
  }
}
```

■ No code found